OFFICIAL

REMARKS

Claims 1-24 and 26-30 are currently pending in the subject application and are presently under consideration.

Applicants' representative notes with appreciation the indication that claims 17 and 19 contain allowable subject matter and would be allowable if rewritten in independent form including all limitations of the base claim and any intervening claims. Nevertheless, it is believed such amendments are not necessary in view of the below-noted novel aspects of the invention as recited in the independent claims *vis-à-vis* the cited art. However, applicant's representative reserves the option to recast claims 17 and 19 in independent form at a later date if necessary.

Favorable reconsideration of the subject patent application is respectfully requested in view of the comments below.

I. Rejection of Claims 3-15, 22, and 23 Under 35 U.S.C. § 112

Claims 3-15, 22, and 23 stand rejected under 35 U.S.C. § 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicants' regard as the invention. Withdrawal of this rejection is respectfully requested for at least the following reasons.

Applicants' representative traverses the Examiner's assertion that applicants' employment of the term "loop" is repugnant to the usual meaning thereof. To clarify, one aspect of the subject invention relates to a method of generating code from a defined mapping between a source schema and a target schema. Accordingly, "loop" should be interpreted relative to the manner in which such term is employed in the computer programming and/or coding arts. In computer science it is commonly known that a loop is a block of code that repeats until a condition is satisfied (*e.g.*, for, while, until...). For instance, an introductory programming book states that "A loop executes a sequence of statements until a particular condition is true (or false)" (Ivor Horton, "Beginning Visual C++ 6", 1998, Wrox Press Ltd., page 110) (attached hereinafter as Exhibit #1). Claims 3-15, 22, and 23 refer to a source loop path node and a source loop path. The specification specifically states "A loop point or source loop path node is a source tree node for which maxoccurs = *" (page 12, lines 17-18). When maxoccurs = * more than one occurrence of a node is possible (page 12, lines 2-4), thus it is necessary to generate

looping code (*e.g.*, for each node...do something) to ensure that all nodes and node data are properly captured. Accordingly, applicants' use of the term "loop" is consistent within the context of the relevant art, namely computer programming. Thus, this rejection should be withdrawn.

II. Rejection of Claims 1, 20, 21, 23, 25, 26, 29, and 30 Under 35 U.S.C. §102(b)

Claims 1, 20, 21, 23, 25, 26, 29, and 30 stand rejected under 35 U.S.C. §102(b) as being anticipated by Greger Lindén, "Structured Document Transformations," 1997, University of Helsinki, Finland, Series of Publications A, Report A-1997-2 (hereinafter Lindén). Withdrawal of this rejection is respectfully requested for at least the following reasons.

Lindén, fails to disclose, teach, or suggest all recited limitations of the claimed invention.

> For a prior art reference to anticipate, 35 U.S.C. §102 requires that "***each and every element as set forth in the claim is found***, either expressly or inherently described, ***in a single prior art reference.***" *In re Robertson*, 169 F.3d 743, 745, 49 USPQ2d 1949, 1950 (Fed. Cir. 1999) (*quoting Verdegaal Bros., Inc. v. Union Oil Co.*, 814 F.2d 628, 631, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987)) (emphasis added). "***The identical invention must be shown in as complete detail as is contained in the ... claim.***" *Richardson v. Suzuki Motor Co.*, 868 F.2d 1226, 1236, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989) (emphasis added).

As per claims 1, 20, 29, and 30, Lindén fails to disclose, teach, or suggest: (i) ***determining source node dependencies*** for a target node by ***tracing from the target node through the mapping*** to the source schema, (ii) matching hierarchy by ***generating a hierarchy match list*** for the target node, and (iii) generating code according to the ***hierarchy match list*** as recited in the subject claims. Lindén teaches a method of specifying source and target relationship rules *via* production group associations and symbol associations, which form a mapping. However such aspect of Lindén (let alone any other aspect thereof) does not anticipate or suggest the recited claim limitation of determining source node dependencies for the target node by tracing from the target node through the mapping to the source schema. Furthermore, the mere fact that a target parse tree is created by a source-to-target mapper (as pointed out by the Examiner), does not disclose, teach, or suggest generating a hierarchy match list for the target node. Necessarily, Lindén also fails to disclose generating code according to said hierarchy

match list. Accordingly, it is readily apparent that claims 1, 20 (21, 23 and 26 which depend there from), 29, and 30 should be allowable as Lindén fails to disclose each and every element as set forth in these claims.

The rejection of claim 25 in this Office Action is believed to be the result of a typographical error, as this claim was canceled in applicants' previous Reply. Accordingly, this rejection is moot.

In view of the above comments, it is readily apparent that Lindén does not teach or suggest applicants' invention as recited in the subject claims, and this rejection should be withdrawn.

III. Rejection of Claims 2 and 16 under 35 U.S.C. §103(a)

Claims 2 and 16 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Lindén in view of Alfred V. Aho, *et al.* "Compilers" Principles, Techniques, and Tools," 1986, Addison-Wesley (hereinafter Aho, *et al*). Withdrawal of this rejection is respectfully requested for at least the following reason.

Claims 2 and 16 depend from independent claim 1, and Aho, *et al.* does not make up for the aforementioned deficiencies of Lindén with respect to claim 1. Accordingly, this rejection should be withdrawn.

IV. Rejection of Claims 18 and 28 under 35 U.S.C. § 103(a)

Claims 18 and 28 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Lindén in view of Alberto Bellina, "XmlTool documentation," 21 January 2003 (hereinafter Bellina). Withdrawal of this rejection is respectfully requested for at least the following reason.

Claims 18 and 28 respectfully depend from independent claims 1 and 20. Bellina does not make up for the deficiencies of Lindén with respect to these independent claims. Therefore, this rejection should be withdrawn.

V. Rej ction of Claim 27 under 35 U.S.C. §103(a)

Claim 27 stands rejected under 35 U.S.C. §103(a) as being unpatentable over Lindén. Withdrawal of this rejection is respectfully requested for at least the following reasons. Claim 27 depends from independent claim 20, and therefore is allowable over Lindén for at least the same reasons noted above in connection with claim 20.

VII. Conclusion

The present application is believed to be in condition for allowance in view of the above comments. A prompt action to such end is earnestly solicited.

In the event any fees are due in connection with this document, the Commissioner is authorized to charge those fees to Deposit Account No. 50-1063.

Should the Examiner believe a telephone interview would be helpful to expedite favorable prosecution, the Examiner is invited to contact applicant's undersigned representative at the telephone number listed below.

Respectfully submitted,

AMIN & TUROCY, LLP

Himanshu S. Amin
Reg. No. 40,894

AMIN & TUROCY, LLP
24TH Floor, National City Center
1900 E. 9TH Street
Cleveland, Ohio 44114
Telephone (216) 696-8730
Facsimile (216) 696-8731

EXHIBIT #1

Beginning Visual C++ 6

Ivor Horton

Wrox Press Ltd.

FYI

Repeating a Block of Statements

The ability to repeat a group of statements is fundamental to most applications. Without this ability, an organization would need to modify the payroll program every time an extra employee was hired, and you would need to reload Tetris every time you wanted to play another game. So let's first understand how a loop works.

What is a Loop?

A loop executes a sequence of statements until a particular condition is true (or false). We can actually write a loop with the C++ statements that we have met so far. We just need an `if` and the dreaded `goto`. Look at this example:

```
// Ex3_07.cpp
// Creating a loop with an if and a goto
#include <iostream>

using namespace std;

int main()
{
  int i = 0, sum = 0;
  const int max = 10;

  i = 1;
loop:
  sum += i;                    // Add current value of i to sum
  ++i;
  if(i <= max)
    goto loop;                 // Go back to loop until i > 11

  cout << endl
       << "sum = " << sum
       << endl;
  cout << endl
       << "i = " << i
       << endl;
  return 0;
}
```

This example accumulates the sum of integers from 1 to 10. The first time through the sequence of statements, `i` is 1 and is added to `sum` which starts out as zero. In the `if`, `i` is incremented to 2 and, as long as it is less than or equal to `max`, the unconditional branch to `loop` occurs and the value of `i`, now 2, is added to `sum`. This continues with `i` being incremented and added to `sum` each time, until finally, when `i` is incremented to 11 in the `if`, the branch back will not be executed. If you run this example, you will get this output:

This shows quite clearly how the loop works. However, it uses a `goto` and introduces a label into our program, both of which are things we should avoid if possible. We can achieve the same thing, and more, with the next statement which is specifically for writing a loop.

TRY IT OUT - Using the for Loop

We can rewrite the last code fragment as a working example using what is known as a `for` loop:

```
// Ex3_08.cpp
// Summing integers with a for loop
#include <iostream>

using namespace std;

int main()
{
   int i = 0, sum = 0;
   const int max = 10;

   for(i = 1; i <= max; i++)          // Loop specification
      sum += i;                       // Loop statement

   cout << endl
        << "sum = " << sum
        << endl
        << "i = " << i
        << endl;
   return 0;
}
```

How It Works

If you compile and run this, you will get exactly the same output as the previous example, but the code is much simpler here. The conditions determining the operation of the loop appear in parentheses after the keyword `for`. There are three expressions that appear within the parentheses:

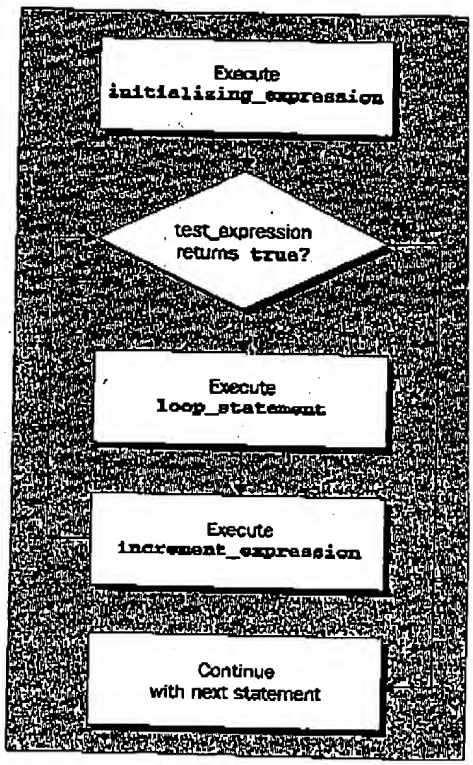
- The first sets `i` to 1
- The second determines that the loop statement on the following line is executed as long as `i` is less than or equal to `max`
- The third increments `i` each iteration

Actually, this loop is not *exactly* the same as the version in `Ex3_ 7.cpp`. You can demonstrate this if your set the value of `max` to 0 in both programs and run them again. Then, you will find that the value of `sum` is 1 in `Ex3_07.cpp` and 0 in `Ex3_08.cpp`, and the value of `i` differs too. The reason for this is that the `if` version of the program always executes the loop at least once, since we don't check the condition until the end. The `for` loop doesn't do this because the condition is actually checked at the beginning.

The general form of the `for` loop is:

```
for (initializing_expression; test_expression; increment_expression)
loop_statement;
```

Of course, `loop_statement` can be a block between braces. The sequence of events in executing the `for` loop is shown here:



As we have said, the loop statement shown in the diagram can also be a block of statements. The expressions controlling the `for` loop are very flexible. You can even put multiple expressions for each, separated by the comma operator. This gives you a lot of scope in applying the `for` loop.

Variations on the for Loop

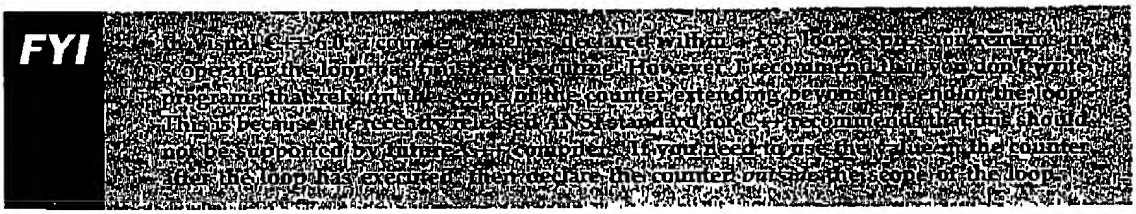
Most of the time, the expressions in a `for` loop are used in a fairly standard way: the first for initializing one or more loop counters, the second to test if the loop should continue, and the third to increment or decrement one or more loop counters. However, you are not obliged to use these expressions in this way and quite a few variations are possible.

The initialization expression in a `for` loop can also include a declaration for a loop variable. Using our previous example, we could have written the loop to include the declaration for the loop counter `i`:

```
for(int i = 1; i <= max; i++)          // Loop specification
  sum += i;                            // Loop statement
```

Naturally, the original declaration for `i` would need to be omitted in the program. If you make this change to the last example, you will find that it runs exactly as before, but there is something odd about this. A loop has a scope which extends from the `for` expression to the end of the body of the loop, which of course can be a block of code between braces, as well as just a single statement. The counter `i` is now declared within the loop scope, but we are still able to refer to it in the output statement, which is outside the scope of `i`. This is because a special extension has been allowed for loop counters to extend their scope to the scope enclosing the loop.

FYI



You can also omit the initialization expression altogether. If we initialize `i` appropriately in the declaration, we can write the loop as:

```
int i = 1;
for( ; i <= max; i++)                  // Loop specification
  sum += i;                            // Loop statement
```

You still need the semicolon that separates the initialization expression from the test condition for the loop. In fact, both semicolons must be in place. If you omit the first semicolon, the compiler will be unable to decide which expression has been omitted.

This flexibility also applies to the contents of the increment expression. For example, we can place the loop statement in the last example inside the increment expression — the loop becomes:

```
for(int i = 1; i <= max; sum += i++);  // The whole loop
```

We still need the semicolon after the closing parentheses, to indicate that the loop statement is now empty. If you omit this, the statement immediately following this line will be interpreted as the loop statement.